
# COMP90024 2024 SEMESTER 1 ASSIGNMENT 2 REPORT

## Cluster and Cloud Computing

**Team 41**
Jiajun Li (1132688)
Luxi Bai (1527822)
Qingze Wang (1528654)
Wenxin Zhu (1136510)
Ze Pang (955698)

**May 2024**
Link to Github repo
Link to Youtube Video

# Contents

# 1 Introduction

The development of technology and breakthroughs in artificial intelligence have revolutionized our lives. The integration of cloud computing, digital science, AI, telecommunications, and social media has not only facilitated our daily living but also sparked numerous societal issues. Social platforms like Twitter and Mastodon provide a venue for the public to express their attitudes, emotions, and preferences about various topical issues in real-time. Similarly, structured data repositories like the Spatial Urban Data Observatory (SUDO) serve as powerful tools for analyzing social phenomena.

This project aims to harness these data integration platforms and tools—Twitter, Mastodon, and SUDO—to deeply explore the impact of AI and data science on people's lives. Specific case studies include the public's emotional attitudes towards Artificial Intelligence (AI), Electric Vehicles (EVs), telecommunications, and 5G, as well as how these sectors relate to people's emotion level.

## 1.1 Motivation

In this project, we primarily focus on investigating the following four major scenarios:

The first scenario examines the impact of breakthroughs in AI on human society. Although the concept of AI has been discussed for many years, the most significant advancements in recent times have been the emergence of large AI models, particularly ChatGPT. Therefore, we use data from 2021 to 2024 to analyze the frequency of AI-related topics on major social media platforms like Twitter and Mastodon, exploring the public's interest in AI. We visually represent the most commonly mentioned issues related to these topics through word clouds. Additionally, we study the distribution of public attitudes towards AI on these social media platforms to analyze how AI's emergence has influenced human life.

The second scenario focuses on the study of new transportation technologies—electric vehicles (EVs). Given the global energy crunch, the development of new energy vehicles like electric cars has surged rapidly, as they are essential for mobility. EVs, particularly those integrated with artificial intelligence, have evolved swiftly, with companies like Tesla and BYD leading the charge as pioneers in this space. Our research centers on Tesla and BYD, examining public attitudes towards EVs on social media platforms such as Twitter and Mastodon. Additionally, we analyze the distribution of vehicles from various companies across different cities to gauge market penetration and consumer preferences.

Our third scenario focuses on telecommunications. In societal life, people's communication relies heavily on internet access and telephony, both of which require services provided by telecom operators. Therefore, to study the impact of telecom companies on people's lives, we selected the three most prominent telecom companies in Australia—Telstra, Optus, and Vodafone—as our subjects of study. We compared public discussions and attitudes toward these companies on social media platforms like Twitter and Mastodon, and also compared their popularity. Since these companies offer different telephony and internet services, analyzing this data helps us understand public preferences for various services.

The fourth scenario explores the impact of 5G technology on people's lives. After examining attitudes towards various telecom companies, we delved deeper into the effects of the telecom companies' primary product—5G services—on daily life. Today, life is inseparable from the internet,

and high-speed network communication quality is a state that everyone strives for. The attitudes towards this service displayed on social media also reflect, to some extent, whether the technological advancements meet people's expectations.

By conducting an in-depth analysis of the above four scenarios, we aim to understand the attitudes of the Australian people towards technological development and hope to gain beneficial recommendations for future technological advancement.

## 1.2 Objectives

To achieve the aforementioned objectives, our project primarily utilizes the IaaS platform provided by MRC to construct a comprehensive cloud computing solution. This solution aims to efficiently utilize cloud computing resources for project deployment, effectively using, analyzing, and visualizing the data gathered from social media regarding the impacts of developments in the fields of AI and EV, etc. in Australia.

Our cloud computing solution comprises the following key components:

- **Python**: We utilize Flask, a Python web framework, as our development framework for writing backend code to interact with databases, process data, and facilitate communication.
- **Elasticsearch (ES)**: ES is a highly scalable open-source full-text search and analytics engine that enables rapid, real-time big data analytics. Therefore, our project uses ES to store and manage data, simplifying subsequent data analyses. Additionally, we use Kibana as the visualization platform for ES, enhancing ease of management and operations.
- **Melbourne Research Cloud (MRC)**: Our solution is implemented via the Melbourne Research Cloud (MRC), which offers a free on-demand computing resources to researchers at the University of Melbourne. MRC uses a private cloud deployment model and provides Infrastructure-as-a-Service (IaaS) cloud computing service, this service makes it easy for researchers to quickly access scalable computational power as their research grows, without the overhead of spending precious time and money setting up their own computing environment [1].
- **Kubernetes**: To achieve our cloud computing solution, we employ Kubernetes as a container orchestration tool to automate the deployment, scaling, and management of containerized applications.
- **Fission**: Fission offers robust function management capabilities, not only easy to install but also capable of requesting functions via flexible routing and automatically scaling functions based on load. We use Fission as the FaaS (Function-as-a-Service) framework within Kubernetes for deploying, scaling, and managing functions.
- **Jupyter Notebook**: This tool is versatile, allowing the creation and sharing of code, equations, visualizations, and machine learning applications. We use Jupyter Notebook as the front-end interactive tool to display results from data processing.
- **Kafka**: Kafka is used for building robust data streaming applications, known for its high throughput, scalability, durability, and fault tolerance. We utilize Kafka to manage data flow processes within our projects.

- **MPI (Message Passing Interface)**: MPI is employed for parallel computing to handle the extensive data involved in our projects. Serial processing is inefficient for such scenarios, making MPI an ideal choice for efficient data processing on a large scale.

By integrating these technologies, we aim to provide a scalable and efficient service capable of processing, analyzing, and presenting large-scale social media data. This service is designed to support our research objectives, enabling us to derive meaningful and helpful results.

### 1.3 Team Roles

Every member of our team has a clearly defined role to ensure the smooth progress of the project. Below are the team members and their respective responsibilities:

- **Jiajun Li (Data Processing & Scenario Analysis)**: Jiajun is primarily responsible for all data processing tasks, including data cleansing, data format transformation, and providing preprocessed data for analysis. This work is crucial for the smooth progression of the entire project, as the quality of subsequent analyses depends on the quality of the original data. Additionally, he is also responsible for data analysis related to 5G scenarios.

- **Luxi Bai (Elasticsearch & Front-end)**: Luxi is in charge of managing and setting up the ES (Elasticsearch) environment. She has primarily completed the initial setup of the ES cluster, including data storage, updates, deletions, and subsequent database management tasks. Additionally, she has also been responsible for writing some of the frontend code for Jupyter Notebook.

- **Qingze Wang (Elasticsearch & Fission)**: Qingze is mainly responsible for managing the ES cluster, including error recovery and data interaction. Furthermore, he is also in charge of deploying the Fission cluster and managing functions, including debugging all functions through four rounds of testing.

- **Wenxin Zhu (Scenario Analysis & Front-end)**: Wenxin focuses on scenario analysis, using data retrieved from ES to analyze scenarios such as AI, EV, and Telecommunication. Additionally, she is responsible for writing some of the front-end code for Jupyter Notebook modules.

- **Ze Pang (Fission & Back-end)**: Ze primarily handles the deployment of the Fission cluster and the writing of functions, crafting relevant Python backend code based on the data retrieval requirements of different scenarios, and managing functions through Fission.

### 1.4 Report Structure

This report begins with a brief introduction to the project's research content and the technologies used in its architecture. It then details the required environment and basic configurations in the User Guide section to facilitate the replication of the project by readers. The third part highlights the architecture and deployment of the entire cloud computing solution, which represents the core technical aspect of the project. The fourth section describes the project's data processing, preparing essential data for subsequent scenario analysis. The fifth part introduces the four main scenarios studied in this project, providing a detailed analysis of the processed data and presenting the corresponding results. The sixth part briefly outlines our tests on Fission functions. The seventh section discusses the solutions to some errors encountered during the project implementation. The

eighth part summarizes the limitations of our project and suggests possible improvements for the future. The final two sections comprise the project conclusion and appendices.

# 2 User Guide

This section offers a concise user guide that outlines the steps to replicate our cloud computing solution. Detailed instructions are as follows:

## 2.1 Source Code

The project code can be cloned locally from the provided GitHub link. Our project source repository is `https://github.com/Flashfooox/CCC-A2-team41`.

## 2.2 Deployed Environments and Packages

- Ensure OpenStack client version 6.3.x is installed, along with the following modules: python-cinderclient, python-keystoneclient, python-magnumclient, python-neutronclient, python-novaclient, and python-octaviaclient.
- Install JQ version 1.6.x for processing JSON data.
- Install Kubectl version 1.26.8 for managing Kubernetes clusters.
- Connect to the campus network if on-campus, or use the UniMelb Student VPN if off-campus.
- **Elasticsearch**
  - ES_VERSION = "8.5.1"
  - replicas = 2
  - Index:
    - incomepsnl: median total personal income weekly SUDO data in Australia
    - twitter-en-geo: English Twitter data with location information
    - mastodon-en: English Mastodon data that harvested with Fission
- **Fission**
  - Required python environment and package are packaged into a Dockerfile.
  - This method eliminates the need to zip the script folder and create a package.
  - Build docker image using Dockerfile and push it to docker hub:

    ```
    docker build -t cccteam41/fission-python-3.9-env:latest .
    docker push cccteam41/fission-python-3.9-env:latest
    ```

  - Create customized environment with docker hub image:

    ```
    fission env create --spec \
        --name env-python3 --image cccteam41/fission-python-3.9-env:latest
    ```

  - Create function by using customized environment:

    ```
    fission function create --spec \
        --name myfunction --env env-python3 --code myfunction.py
    ```

- Create timetrigger that allows harvester to restart every 5 minutes:

```
fission timer create --spec --name myfunction-timer --function myfunction \
    --cron "@every 5m"
```

- Create route that allows functions to be triggered by HTTP routes (URL paths):

```
fission route create --spec --name myfunction-route --url "/myfunction/{param}" \
    --method POST --createeingress --function myfunction
```

- Create mqtrigger that allows automatically scale dataprocessor based on the volume of messages in the Kafka topic-data:

```
fission mqtrigger create --spec --name dataprocessor --function dataprocessor \
    --mqtype kafka --mqtkind keda --topic topic-data --maxretries 3 \
    --resptopic topic-observations --errortopic topic-errors \
    --metadata bootstrapServers=mycluster-kafka-bootstrap.kafka.svc:9092 \
    --metadata consumerGroup=my-group --cooldownperiod=30 --pollinginterval=5
```

- Apply the specs to the cluster:

```
fission spec apply --specdir fission/specs --wait
```

- **Kafka**
  - Five topics are deployed in cluster "mycluster" through YAML files:
    - topic-twitter: manage Twitter data streaming or processing tasks
    - topic-mastodon: manage real-time Mastodon data streaming or processing tasks
    - topic-income: manage SUDO income data streaming or processing tasks
    - topic-errors: where error messages from the function execution are sent
    - topic-observations: where the function's response messages are sent
  - Create Kafka cluster with YAML file:

```
kubectl apply -f ./kafka/topics/topic.yaml --namespace kafka
```

  - Create Kafka topic with YAML file:

```
kubectl apply -f ./kafka/topics/topic-data.yaml --namespace kafka
```

### 2.3 Frontend

The Frontend of this project is implemented using Jupyter Notebook. The primary purpose of the frontend is to interact with a backend server to fetch and display data. The following Python code snippets demonstrate how to retrieve data from the server using HTTP requests.

First, run the AsyncFissionClient.py use the command:

```
python AsyncFissionClient.py
```

Figure 1 shows the output after running this command. This python script is used for bulk data retrieval from a specified server, processing multiple requests in a non-blocking fashion to optimize performance. The log entries indicate that asynchronous functions are employed to manage multiple HTTP requests simultaneously for efficient data fetching and processing. Asynchronous

```
2024-05-28 20:13:11,631 - INFO - Requesting URL: http://localhost:9090/search/twitter/keyword/Telstra
2024-05-28 20:13:11,633 - INFO - Requesting URL: http://localhost:9090/search/mastodon/keyword/Telstra
2024-05-28 20:13:11,634 - INFO - Requesting URL: http://localhost:9090/wordcloud/twitter/keyword/Telstra
2024-05-28 20:13:11,635 - INFO - Requesting URL: http://localhost:9090/wordcloud/mastodon/keyword/Telstra
2024-05-28 20:13:11,636 - INFO - Requesting URL: http://localhost:9090/retrieve/incomepsnl
2024-05-28 20:13:13,988 - INFO - Data saved to es_data/search_twitter_Telstra.json
2024-05-28 20:13:14,605 - INFO - Data saved to es_data/wordcloud_twitter_Telstra.json
2024-05-28 20:13:14,916 - INFO - Data saved to es_data/wordcloud_mastodon_Telstra.json
2024-05-28 20:13:16,716 - INFO - Data saved to es_data/retrieve_incomepsnl.json
2024-05-28 20:13:18,181 - INFO - Data saved to es_data/search_mastodon_Telstra.json
2024-05-28 20:13:18,181 - INFO - Request successful, status code: 200
2024-05-28 20:13:18,182 - INFO - Request successful, status code: 200
2024-05-28 20:13:18,182 - INFO - Request successful, status code: 200
2024-05-28 20:13:18,182 - INFO - Request successful, status code: 200
2024-05-28 20:13:18,182 - INFO - Request successful, status code: 200
```

Figure 1: AsyncFissionClient.py output

programming enhances system performance and scalability by allowing these tasks to occur concurrently instead of one after the other. This approach keeps the execution thread active, optimizing the use of resources, shortening the total operation time, and improving the system's ability to handle high loads by executing other tasks while waiting for responses. The data fetched is saved in JSON format within a designated directory, enabling further analysis or processing.

Next, execute all Jupyter notebooks for the frontend to generate visualizations of the plots.

# 3 System Architecture and Design

## 3.1 Allocated Resources

Our team received a total of 6 instances, 11 virtual CPUs, and a 700-gigabyte volume allocation. In our system's setup, we opted to distribute these resources across 5 instances, each equipped with 9 VCPUs, and 40.5 gigabytes of RAM as depicted in Figure 2.

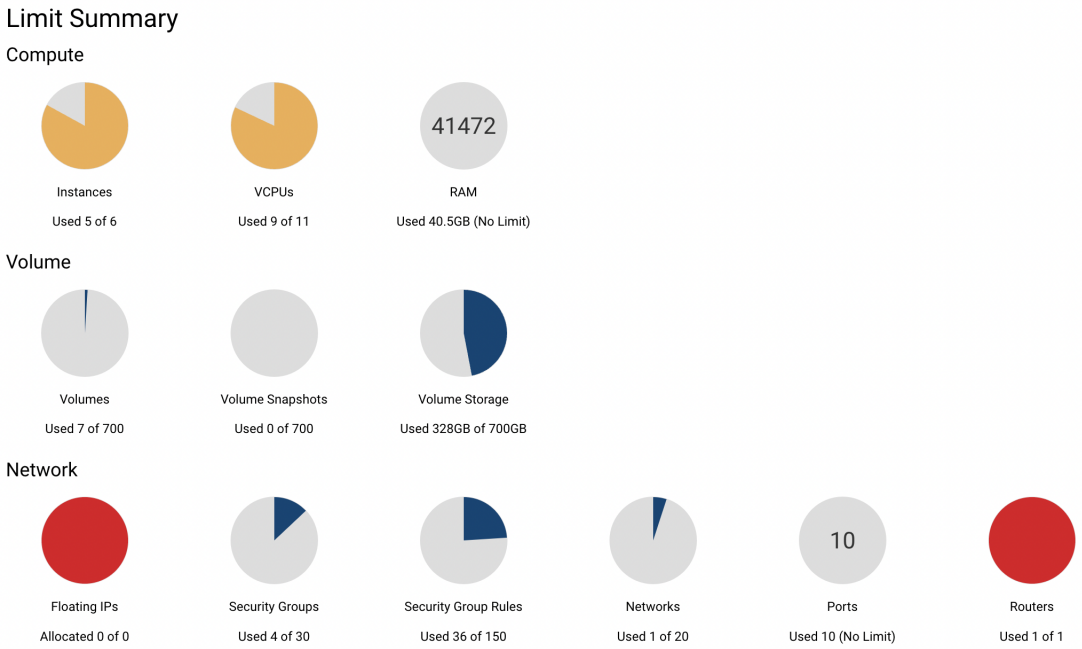


Figure 2: Instance Usage

### 3.2 System Structure

The system is implemented in the Melbourne Research Cloud (MRC) environment utilizing Kubernetes clustering and related tools to achieve secure, efficient, and scalable service deployment and management. The following system architecture diagram shows how, through the collaborative work of the components, the system is able to meet the access needs of multiple users and provide flexible scaling and efficient resource management while maintaining security.

The architecture ensures secure access for developers through an SSH tunnel and users via RESTful requests. The bastion host serves as a secure entry point for managing the cluster. The project structure is allocated as follows:

- backend/
    - fission/
        - functions/: Folder that stored all fission functions
        - specs/: Folder that stored all fission configuration YAML files
        - Dockerfile: Docker image that used to create fission python environment
    - kafka/
        - topics/: Folder that contains topic configuration YAML files
        - kafka-cluster.yaml: File that used to create Kafka cluster
- database/
    - schema/: Folder that contains JSON schema files which used to create the Elasticsearch indexes.
    - sudo-download/: Folder that contains download CSV file from SUDO
    - twitter-upload/: Folder that contains scripts to download Twitter data from Spartan with MPI4PY and downloaded JSON data
    - uploader/: Folder that contains scripts to upload JSON data to Elasticsearch with Fission enqueue route.
- frontend/
    - es_data/: Folder that stored all the fetched data from Elasticsearch.
    - plots/: Stored all plots generated by jupyter notebooks.
    - shapefile/: SA4 shapefile that used to plot maps.
    - AsyncFissionClient.py: Python script that fetch data asynchronously for frontend
    - DataAnalyzer.py: Python script designed for analyzing and visualizing data retrieved from the server.
    - ScenarioAnalyzer.py: Python script for creating various types of plots to analyze and compare data across different scenarios.
    - frontend_{scenario}.ipynb: 4 notebooks that related to scenarios AI, EV, Telecommunications, 5G
    - api_test.ipynb: Code for test whether the data can be obtained successfully
- tests/: Folder that contains 4 iterations to test connection and functions with Elasticsearch.

- 1st-iteration/
- 2nd-iteration/
- 3rd-iteration/
- 4th-iteration/

### 3.3 System Architecture

Figure 3 describes a system architecture primarily based on Kubernetes (K8s) for deploying and managing containerized applications. Here's an explanation of each component and how they interact:

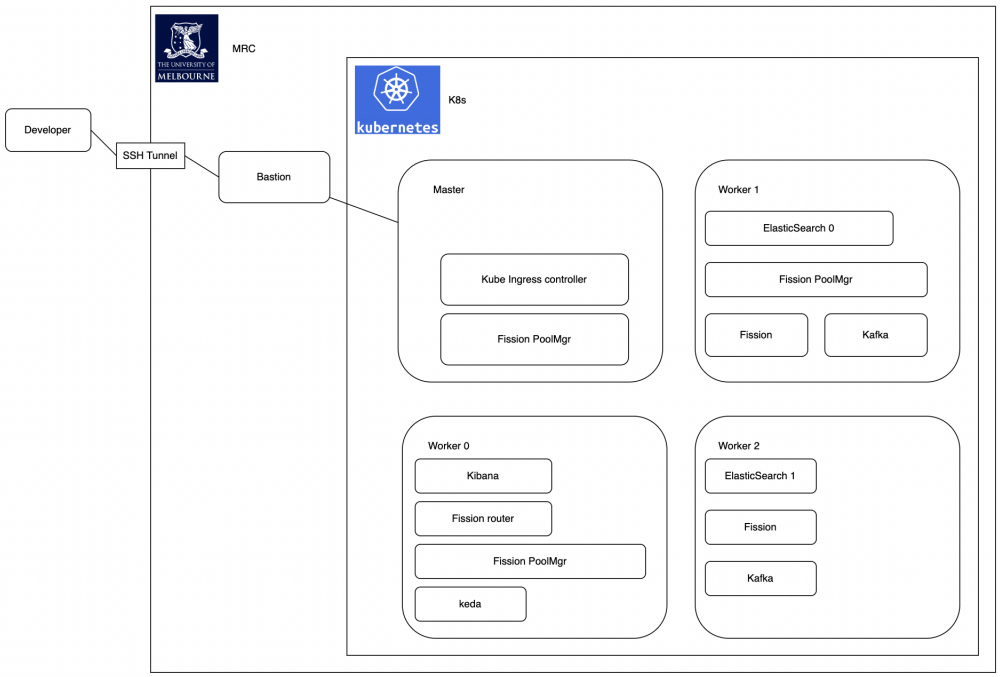


Figure 3: Infrastructure Overview

- Developer: The individual or team responsible for developing and maintaining the system.
- SSH Tunnel: A secure channel through which the developer can access the system.
- Bastion: A server that acts as a gateway between the developer and the Kubernetes cluster, providing an additional layer of security.
- Kubernetes Cluster (K8s):
  - Master Node:
    - Kube Ingress controller: It manages external access to the services in the cluster. When an external user wants to use an internal feature, they need to make a Restful request to the appropriate route, and the Ingress controller will load-balance and provide the registered Fission feature.

    - Fission PoolMgr: It oversees the function pools in the Fission framework. It reduces the resources spent on starting and destroying pods by keeping the right number of pods for running functions.
  - Worker 0:
    - Kibana: It provides visualization capabilities for Elasticsearch data
    - Fission router: It handles function routing of fission.
    - Fission PoolMgr: Manages function instances for Fission.
    - Keda (Kubernetes Event-Driven Autoscaling).
  - Worker 1:
    - Elasticsearch 0: An Elasticsearch replica that is responsible for storing the data shard and sharing the query pressure.
    - Fission PoolMgr
    - Fission
    - Kafka: A distributed event streaming platform.
  - Worker 2:
    - Elasticsearch 1
    - Fission
    - Kafka

Figure 4 shows the workflow of the system:

- Data Sources:
  - SUDO: Source for median total personal income weekly data.
  - Twitter: Utilize twitter-100gb.csv from Spartan and processes the Twitter data using MPI for parallel processing.
  - Mastodon: Utilize 2 mastodon servers to harvest streaming data.
- MPI4PY: A Python binding for Message Passing Interface (MPI) used for parallel processing.
- Kubernetes Cluster (K8s):
  - Fission Functions:
    - mastodonharvester1 and mastodonharvester2: Functions to harvest stream data from two different Mastodon servers.
    - enqueue: Function to enqueue processed data into Kafka topics.
    - incomeprocessor, twitterprocessor, mastodonprocessor: Functions to process different types of data (income, Twitter, Mastodon).
    - search: Function to perform searches on the processed data for frontend.
    - wordcloud: Function to provide words and their frequency for frontend.
    - retrieve: Function to retrieve data for the front-end like income data.
    - addobservations: Function to subscribe to the Kafka topic and add observations to Elasticsearch.
  - Kafka Topics: income, twitter, mastodon, observations: Different Kafka topics to organize streaming data.

  - Elasticsearch: Stores observations from Kafka for efficient searching and retrieval.

- Front-end: Utilize Jupyter Notebooks, Pandas, Matplotlib... for data analysis and visualization.
- User: The end user who interacts with the system to get data and perform analysis.

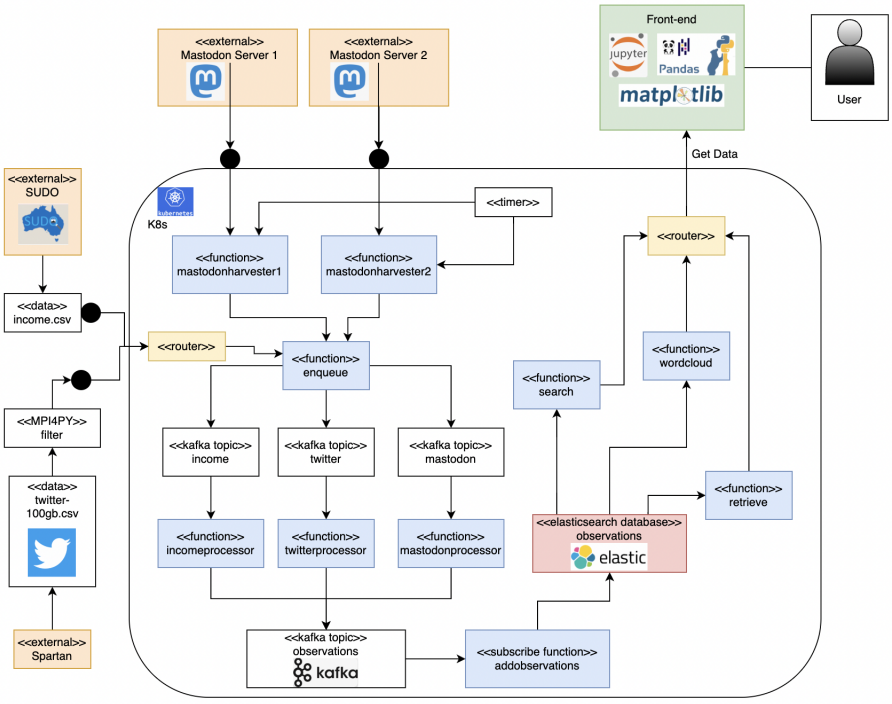


Figure 4: Application Workflow

# 4 Dataset and Wrangling

This section outlines our methods for handling varied datasets sourced from Twitter, Mastodon, and SUDO.

## 4.1 Twitter Data Processing

The original dataset, twitter-100gb.csv, stored on Spartan, comprises structured JSON Twitter data primarily from 2021 to 2022. Due to constraints related to internet bandwidth and local machine memory, it is impractical to download and upload this large file directly to Elasticsearch. To address this, we employed a method to efficiently filter tweets containing geo-information by opening the file in binary mode and utilizing regular expressions. This process also filters out non-English tweets and selects specific attributes for further use. To accelerate the filtering process, we leveraged MPI (Message Passing Interface) to distribute tasks across multiple nodes, reducing the task completion time to approximately one minute on 8 nodes.

The resultant data, now condensed into a manageable 1GB JSON file contains 4 million tweets, was then downloaded. Following this, the data, along with the designated Elasticsearch index name, were uploaded to the Kafka Twitter topic via a Fission enqueue HTTP trigger. The data were subsequently processed using mastodonprocessor.py, where sentiment analysis was performed using Python's nltk.sentiment module, and readability metrics were computed with the readability package. Finally, the processed data were uploaded to the specified Elasticsearch index using addobservations.py. This streamlined approach significantly reduced data size and processing time, enhancing the efficiency of data handling and analysis.

### 4.2 Mastodon Data Processing

Apart from Twitter, Mastodon data is also utilized for social analysis. Mastodon [2] is an open-source social networking platform simailr to Twitter. The data collection involves two Python scripts, mastodonharvester1.py and mastodonharvester2.py, which harvest public data every five minutes from the aus.social and mastodon.au servers, respectively. These scripts utilize a custom listener for the Mastodon API to handle streaming data, which is then uploaded to the Kafka Mastodon topic using the enqueue route. The data is processed by mastodonprocessor.py, a procedure similar to that used for Twitter data. This process involves the removal of HTML tags from the content, and the calculation of various sentiment and readability scores. After processing, the data is uploaded to Elasticsearch using addobservations.py. The total size of the dataset is approximately 1.7 million entries.

### 4.3 Spatial Urban Data Observatory (SUDO)

The SUDO [3] dataset is well-structured and well-formatted, hence no need for further processing. In accordance with our focus scenarios, we utilized one primary dataset: **SA4-P02 Selected Medians and Averages**, which is Median total personal income weekly in Australia.

To generate meaningful maps, we incorporated the Statistical Area Level 4 shapefile provided by the Australian Bureau of Statistics (ABS) [4] for future analyses. Consequently, we generated map plots and summary reports, which were stored on the frontend for quick user access.

There is no scalability design for SUDO data. Given the nature of SUDO data, distinct processing methods are required based on the selected scenario topics.

## 5 Scenarios and Analysis

This application analyzed topics related to technology concerning public sentiment towards AI, electric vehicles (EVs), 5G, and telecommunications, along with how these industries related to different income levels across Australia.

### 5.1 AI

Figure 5 and Figure 6 shows the sentiment distribution about Twitter and Mastodon related to AI, the left pie chart shows that the majority of the tweets (64.4%) have a positive sentiment, while 17.9% are neutral, and 17.7% are negative. This indicates that the general perception of AI on Twitter is predominantly positive. The right pie chart shows 48.8% of the toots are positive, 27.3% are neutral, and 23.9% are negative. This two charts suggests that public opinion and sentiment

towards AI topics was more favorable and positive two years ago, as reflected in tweets from that time period. In contrast, the current sentiment captured from Mastodon shows a relatively less positive view, with a higher proportion of neutral or negative opinions about AI compared to the past tweets data.

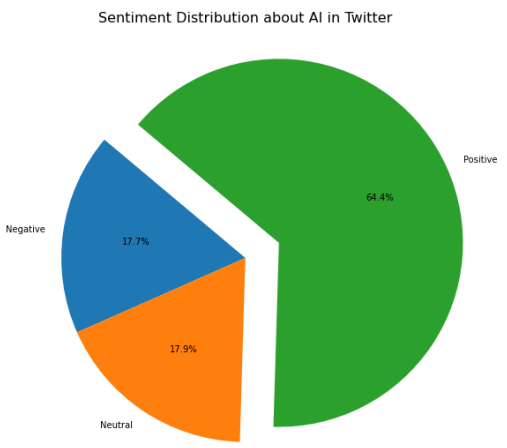


Figure 5: Sentiment about AI in Twitter

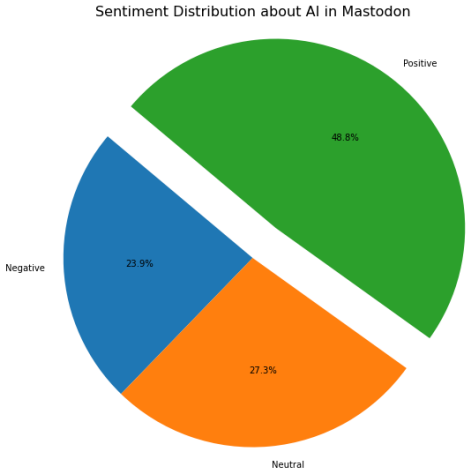


Figure 6: Sentiment about AI in Mastodon

From Figure 7, this line graph displays the frequency of tweets related to AI topics over time. The graph highlights periods of higher activity such as 2021-09, suggesting spikes in interest or events related to AI that prompted increased discussion. Figure 8 presents the frequency of toots related to AI topics over time on Mastodon. Noticeable peaks indicate times when discussions about AI were particularly prominent such as between 2024-04-17 and 2024-04-21, likely due to significant events or announcements in the AI field.

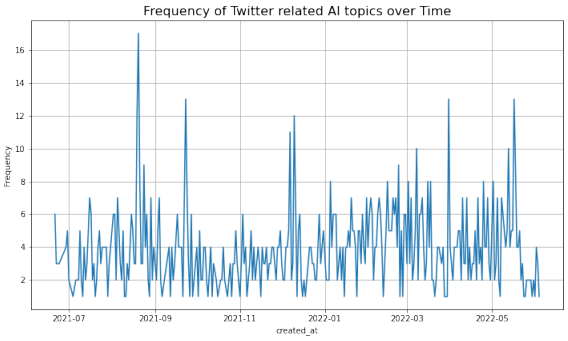


Figure 7: Frequency of Twitter related AI

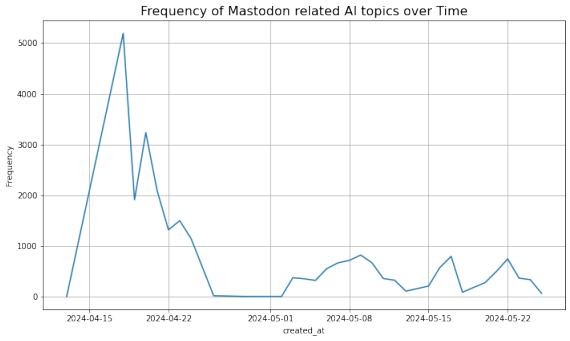


Figure 8: Frequency of Mastodon related AI

Therefore, we can look at the wordcloud of toots in that period from Figure 9, high frequency terms include “chatbot”, “generative AI”, “AI model”, “generativeArt”, “AIArt” and “ChatGPT”. This suggests that there was significant interest and discourse around these emerging technologies and their applications, particularly in the realm of natural language processing (chatbots and language models) and generative art created by AI systems.

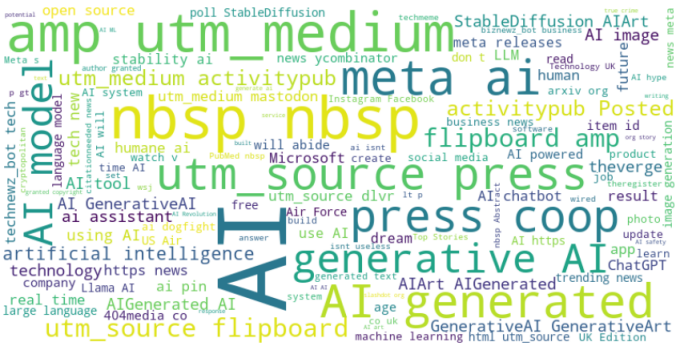


Figure 9: WordCloud for Mastodon about AI

The distribution of median total personal income weekly for SA4 regions in Australia is shown in Figure 10, the upper north-west region has the highest income level, with about $1300 weekly, New South Wales and Victoria also shows relatively high income levels, with about $850 weekly, whereas the Northern Territory and some parts of South Australia show lighter shades, indicating lower median incomes about $600 weekly.

The distribution of AI sentiment is illustrated in Figure 11, where sentiment scores vary from -1 (very negative) to +1 (very positive). Regions colored white indicate no mention of AI topics on Twitter. For regions where AI is mentioned, it is observed that the southern regions and east of Queensland tend to have a positive attitude towards AI, while the northern territory exhibit a negative attitude towards AI.

These two maps suggest a possible correlation where regions with higher median incomes tend to have a more positive sentiment towards AI. This relationship highlights the impact of economic factors on public perceptions of technological advancements. High-income areas, with greater access to resources and education, may be more optimistic about AI's potential, while lower-income areas might be more cautious or critical due to potential economic concerns.

## 5.2 Electric Vehicles (EVs)

Figure 12 shows the sentiment distribution about EV in Twitter and Mastodon, it shows that public sentiment about EVs has remained relatively stable, with neutral sentiment being predominant in both datasets. Positive sentiment occurs more frequently than negative sentiment on Twitter, whereas on Mastodon, positive and negative sentiments are observed at similar frequencies. One possible reason for the difference could be the change in public opinion over time. Since Twitter data is from two years ago, it's possible that public sentiment was more optimistic during that period, resulting in a higher frequency of positive sentiments. In contrast, current data from Mastodon might reflect a more balanced view. Additionally, the different user demographics and platform cultures of Twitter and Mastodon might contribute to the variation in sentiment distributions.

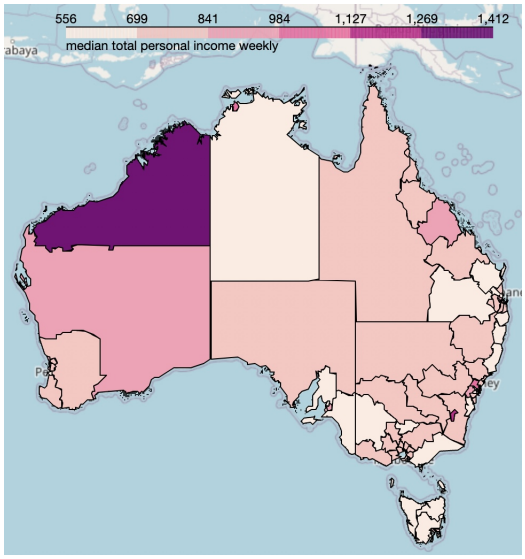


Figure 10: Distribution about income

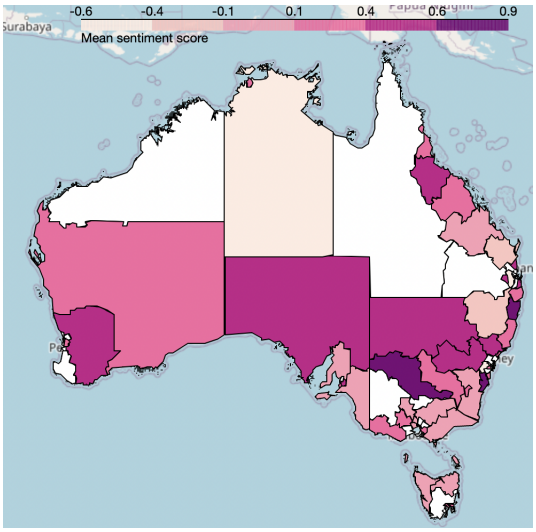


Figure 11: Distribution about AI sentiment

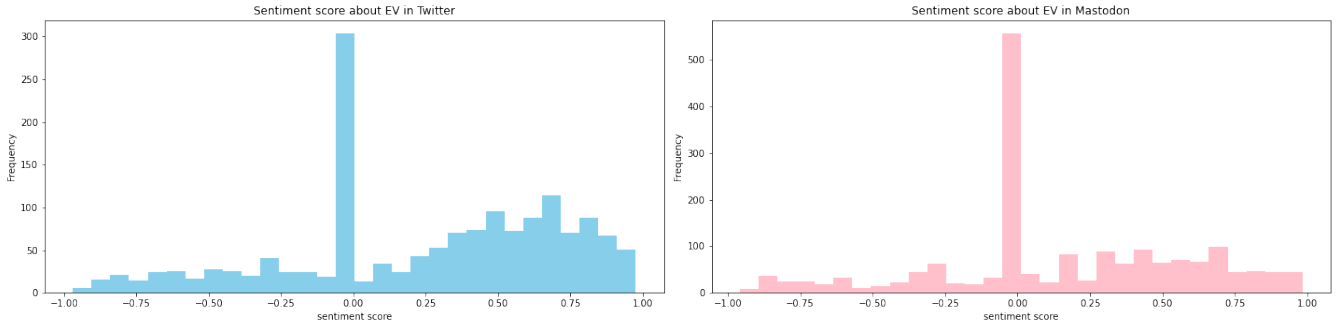


Figure 12: Sentiment Distribution about EV

Next, We examine two popular EV brands: Tesla and BYD.

Figure 13 shows the boxplot comparison about the sentiment Tesla and BYD on Twitter. For positive sentiment: both Tesla and BYD have similar sentiment scores, about 0.55. For neagtive sentiment: Tesla also has a wider range of negative sentiment scores compared to BYD, indicating more polarized opinions about Tesla, and the median sentiment score of BYD (around -0.6) is lower than Tesla (around -0.5).

Figure 14 shows the sentiment distribution in Mastodon. For positive sentiment: Tesla decreased to 0.4 whereas BYD continue to have high median sentiment score about 0.55. For negative Sentiment: Tesla shows lower sentiment score than BYD.

Therefore, we can see that public opinion towards BYD has improved over the past two years. This could be due to several reasons such as BYD may have expanded its market presence and increased its sales globally, leading to greater brand recognition and positive reception.

Figure 15 shows the sentiment distribution about EV in Mastodon, related it to the income distri-

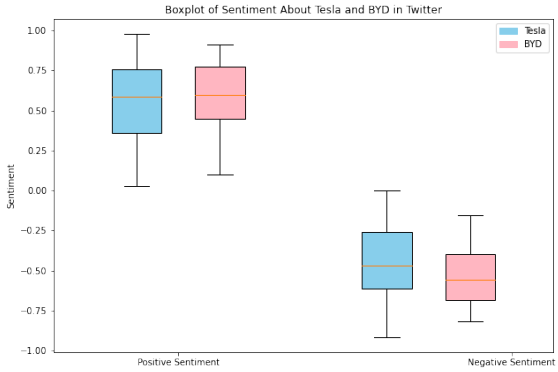


Figure 13: Boxplot of Tesla & BYD in Twitter

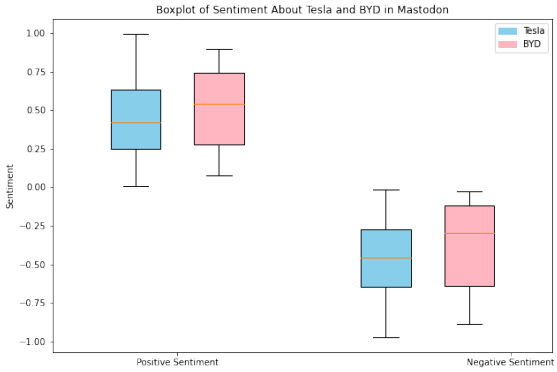


Figure 14: Boxplot of Tesla & BYD in Mastodon

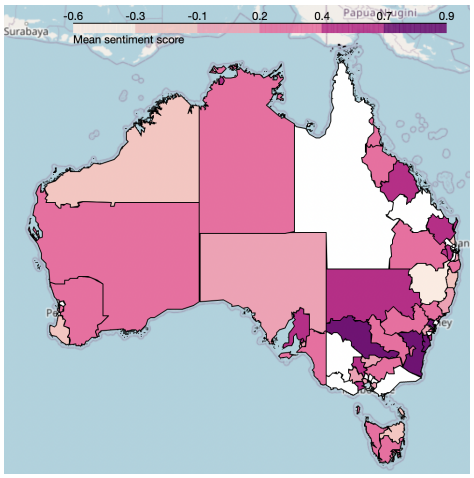


Figure 15: Distribution about EV sentiment

bution in Figure 10, we might demonstrate that there is a trend where regions with higher median incomes tend to have a more positive sentiment towards EVs. However, this relationship is not uniform across all regions, indicating that other factors such as environmental awareness, regional economic activities may play significant roles in shaping public sentiment towards EVs. Higher incomes can make the adoption of new technologies like EVs more feasible, but the sentiment towards EVs also depends on how well these vehicles are supported within the region.

### 5.3 Telecommunication

Figure 16 tracks the sentiment of tweets mentioning Telstra, Optus, and Vodafone from 2021-07 to 2022-05. The sentiment for Telstra and Optus fluctuate significantly, indicating varying user experiences and opinions over time. While Vodafone's sentiment shows less variability than Telstra and Optus, with some moderate peaks and troughs, but generally stays within a closer range around the neutral line (0).

Figure 17 shows the sentiment of toots on Mastodon mentioning the same three companies. The sentiment for Telstra on Mastodon is more negative than on Twitter, with notable negative peaks. Optus shows a relatively more positive trend compared to Twitter, with significant high sentiment

such as 2024-04-21 and 2024-05-17. Vodafone's sentiment on Mastodon is relatively stable and hovers around the neutral line, similar to its trend on Twitter but with fewer fluctuations.

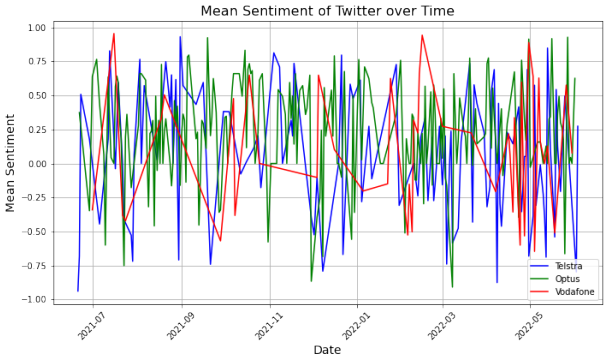


Figure 16: Mean sentiment of Twitter

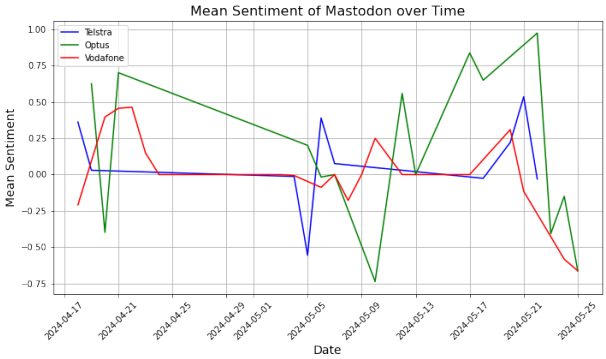


Figure 17: Mean sentiment of Mastodon

Figure 18 illustrates the popularity of Telstra, Optus, and Vodafone. Notably, Optus was very popular on Twitter two years ago, garnering 259 positive tweets (total 478 toots). However, its current popularity on Mastodon is significantly lower, with only 28 toots in total, of which 14 are positive.

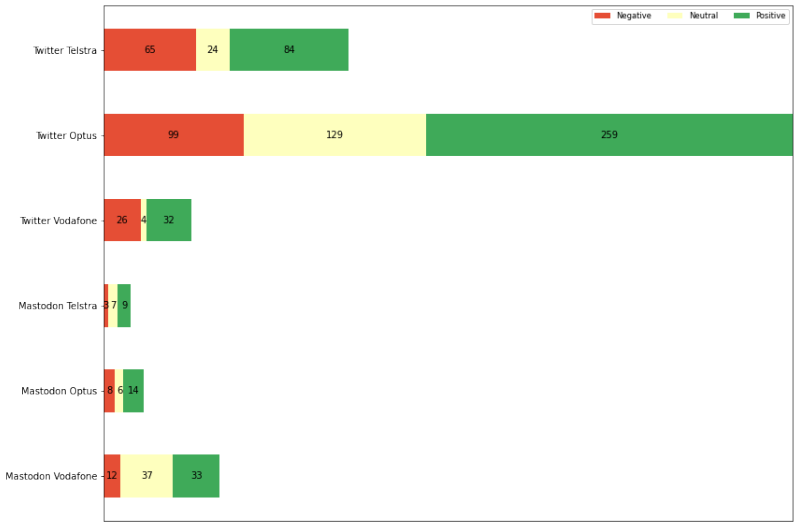


Figure 18: Popularity of Telstra, Optus and Vodafone

Therefore, to understand the shift in attitudes towards Optus, we created word clouds for tweets and toots. In Figure 19, we observe frequent words like "good", "well", "great" and "potential", indicating a more positive sentiment. However, in Figure 20, the high frequency shifts to words like "took minutes", "trouble", "slow" and "issue", reflecting a more negative tone.

To conclude, the popularity and sentiment surrounding Telstra, Optus, and Vodafone reveals intriguing shifts over time, particularly concerning Optus. While it enjoyed significant popularity on Twitter two years ago, its presence on Mastodon has notably dwindled. Such insights can be


Figure 19: Wordcloud of Optus in Twitter

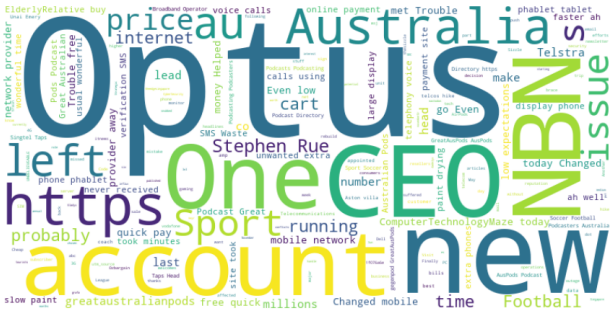


Figure 20: Wordcloud of Optus in Mastodon

valuable for telecommunications companies seeking to adapt their strategies and address evolving consumer sentiments effectively.

## 5.4 5G

In 2019, Telstra announced the deployment of its 5G network in major cities such as Sydney, Melbourne, Brisbane, Adelaide and Perth. In the same year, Optus announced the start of 5G services to consumers. Driven by these leading operators, Australia's 5G network coverage is rapidly increasing. By 2020, Telstra's 5G network had covered 41% of the population, and by 2021, that number surged to 75%. by 2023, Telstra's 5G network coverage surpassed 85%. Figure 21 shows the growth in the number of 5G sites deployed by each operator[5]. Telstra is well ahead of its competitors. Optus and TPG are also working hard to expand their own number of 5G base stations to provide high-speed internet services to more users.

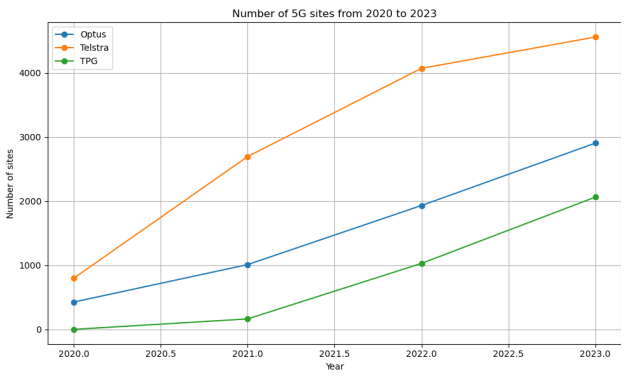


Figure 21: Number of 5G sites from 2020 to 2024

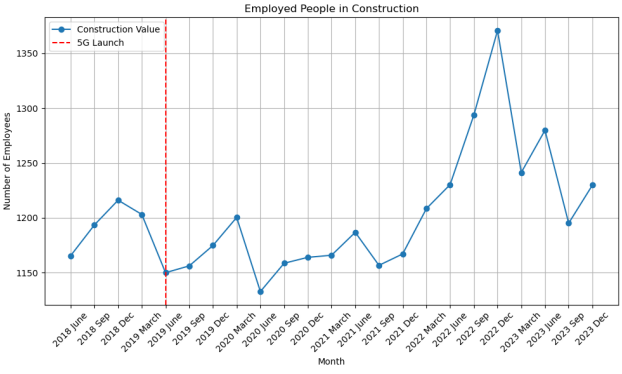


Figure 22: Employed People in construction from 2018 to 2023

The rapid expansion of 5G networks has led to a boom in the construction industry. With the demand for the construction of a large number of 5G base stations, the employment market in the construction industry has grown dramatically. From 2018 to 2023, hiring in the construction industry shows a significant upward trend, especially after the introduction of 5G technology in 2019,

the demand for construction workers soars. In the Figure 22,at the end of 2021, with the gradual improvement of the COVID-19 epidemic (e.g., zero mortality rate), the construction industry once again sees a peak in hiring. Increased investment in infrastructure development by governments and corporations not only drives the construction of 5G base stations, but also provides more employment opportunities for construction workers.

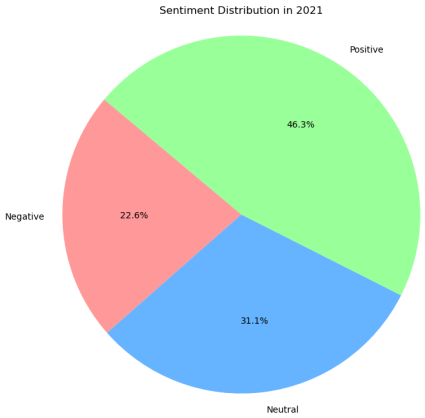


Figure 23: Sentiment Distribution in 2021

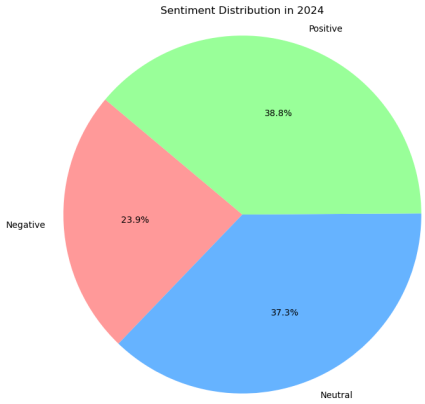


Figure 24: Sentiment Distribution in 2024

5G technology has not only changed the construction industry, but also triggered a wide range of discussions in social opinion. Figure 23 demonstrates that, there is a diversified trend in the attitudes toward 5G technology expressed through social networks (e.g., Twitter) in 2021. 22.6% of the people have a negative attitude toward 5G, 31.1% are neutral, while 46.3% have a positive attitude toward 5G. Figure 24 shows that, with the gradual popularization of 5G technology and the expansion of application scenarios, social opinion has also changed in 2024. More and more people realize the important role of 5G technology in improving the quality of life and promoting economic development, and the acceptance and recognition of 5G has changed. 38.8% of people have a positive attitude towards 5G, 37.3% are neutral, and 23.9% have a negative attitude. The proportion of neutral comments on social networks increased significantly, reflecting a more rational public evaluation of 5G technology.

# 6 Test

## 6.1 Test cases

Our project involved four iterations of testing that focused on how to build and test a function-as-a-service (FaaS) application with a RESTful API in the Fission FaaS framework.

Each iteration of testing corresponds to a different stage in the development process and is designed to verify the implementation and integration of features and ensure that the application works as expected. The following is the main content of each iteration of testing

### 6.2 1st Iteration

Fission function creation: Write basic functions, such as those for Mastodon and Twitter.

Route setup: Create routes for these functions so that they can be accessed through specific URL paths.

Testing: Perform a simple end-to-end test to verify that routes exist, that functions are called correctly, and that they return status code 200.

After the first round of testing, all basic functions have been created and are accessible through these routes. Verify that the HTTP routing settings of the application are correct, and the request can be correctly sent to the corresponding function. A simple end-to-end test verifies that the function can be called correctly and returns the expected HTTP status code (200), ensuring the basic accessibility and functionality of the application.

### 6.3 2nd Iteration

Global parameter Settings: Use Kubernetes ConfigMap to manage global parameters required, such as database URLs, database user name and passwords.

Database integration: Create Elasticsearch database instances and configure indexes and mappings for Mastodon and Twitter.

Test: Verify that Fission function can successfully read the database and return the correct data structure.

After the second iteration of testing, we successfully used ConfigMap to manage the global parameters, such as database connection information, ensuring the secure management of sensitive information. Proper database configuration, including the creation of indexes and maps, ensures that data can be stored and retrieved correctly. It also verifies that the function can interact with the Elasticsearch database, read the data correctly, and return a structured JSON response, which indicates that the application can handle more complex data operations.

### 6.4 3rd Iteration

Test database: Set up a test database environment where data can be emptied and added before testing.

Dealing with side effects: Develop functions that modify the state of the database (such as adding or deleting Mastodons, Twitters, etc.).

CRUD testing: Testing CRUD operations to ensure that creations, updates, deletions, etc. are performed correctly and data is consistent.

The third iteration of testing is to ensure that the test database environment setup is complete and can be reset before each test to ensure the independence and consistency of the test.

Functions capable of modifying the state of the database (create, update, delete) were successfully implemented and tested to verify that they worked as expected.

End-to-end testing confirms that these CRUD operations do not affect the overall stability and performance of the application, ensuring the correct implementation of more complex operations.

End-to-end testing: Ensure that the entire system remains stable during database transactions.

### 6.5 4th Iteration

Code review: Optimize existing code, eliminate duplication, and improve code maintainability and extensibility.

Unit testing: Write unit tests for our new code to ensure that each module is correct when run independently.

Continuous testing: Use end-to-end testing to verify that the entire system still works as expected.

The final test is to ensure that the code review improves the maintainability and extensibility of the application and simplifies the management of complex functionality.

The unit test verifies the independent function of each module after reconfiguration to ensure the correctness of the module level.

Ongoing end-to-end testing verified that the entire application performed as expected after the refactoring, with no degradation in performance or functionality, confirming the success of the refactoring.

## 7 Pros and Cons of Tools

### 7.1 Melbourne Research Cloud

The Melbourne Research Cloud (MRC) is an academic research cloud platform provided by the University of Melbourne that provides high-performance computing resources and large-scale data storage to support the performance of complex computations and data analysis.

#### 7.1.1 Advantages and Disadvantages

- **Advantages**:
  - **High performance computing**: Multi-core CPUs are provided to allow researchers to parallelize large computational tasks and increase computational efficiency. Large memory instances are provided for tasks that require large amounts of data to be processed, ensuring the speed and efficiency of data processing.
  - **Massive Data Storage**: Block Storage provides high-performance block storage solutions for applications that require fast reading and writing of data. File storage facilitates users to share data among multiple computing instances.

- **Ease of Use**: MRC provides user-friendly interfaces and tools that enable researchers to easily manage and utilize computing resources and storage. With intuitive graphical user interfaces such as the Graphical User Interface (GUI), users can manage and monitor computing resources and storage without complex command line operations.
- **Security**: MRC provides strong security mechanisms to ensure the security and privacy of research data. Through authentication and authorization, a strong authentication mechanism is used to ensure that only authorized users can access computing resources and data.
- **Portability**: Since MRC is based on OpenStack, which contains many open source software, the deployment of this project is compatible with any hosting provider that supports the OpenStack platform. This facilitates ease of migration in the future, if ever required.

- **Disadvantages**:
  - **Complexity**: Initial learning and configuration can be complicated for users unfamiliar with cloud computing technologies.
  - **Dependency**: Dependent on the availability of internet connection and cloud services, any network problems or service interruptions may affect the research process.
  - **Responsiveness**: Compared to widely-used public cloud platforms like Amazon AWS and Microsoft Azure, addressing issues on the MRC requires submitting a "ticket" for the cloud administrator to investigate. The response time for these tickets is not guaranteed, which means that if bugs occur on MRC, they could potentially delay development for an unspecified duration.

## 7.2 Kubernetes

Kubernetes is a powerful container orchestration platform used for automating the deployment, scaling, and management of containerized applications.

### 7.2.1 Pros and Cons

- **Pros**:
  - **Automation**: Kubernetes automates the application deployment process, reducing human error and improving deployment efficiency. Moreover, it supports Auto-scaling, which automatically increases or decreases the number of containers based on load, ensuring that applications run efficiently under different loads. When a container or node fails, Kubernetes will automatically re-schedule and restart the container to ensure high availability of the application.
  - **Community Support**: Kubernetes has a large and active community of developers and users, providing a wealth of documentation, tutorials, and examples. A large number of open source tools and plugins can be integrated with Kubernetes to extend its functionality, such as Helm, Prometheus, Istio, and others.

- **Cons**:

- **Complexity**: Learning and using Kubernetes requires some technical background and experience, making the learning curve steeper for beginners and small teams. The configuration and management of Kubernetes involves multiple components and concepts, such as Pod, Service, Deployment, Ingress, etc., which require in-depth understanding and proficiency.
- **Resource Intensive**: Running Kubernetes requires a lot of computing resources, especially in production environments that require multiple nodes, high-availability configurations for master nodes, and monitoring and logging systems. When making configurations, certain resource predictions need to be done to ensure stability.

## 7.3 Fission

Fission is a framework that provides Serverless architecture capabilities for Kubernetes, enabling Functions as a Service (FaaS) on the Kubernetes platform. It allows developers to deploy, manage, and scale functions without having to focus on the underlying server management.

### 7.3.1 Advantages and Disadvantages

- **Advantages**:
  - **Simplified Deployment**: Developers can focus on writing function logic without managing servers. Functions can be deployed quickly, saving a lot of O&M work.
  - **Auto Scaling**: Fission automatically scales up and down function instances based on request volume, ensuring that enough compute resources are available during high loads and saving resources during low loads.
  - **Efficient Use of Resources**: Functions consume resources only when they are called, enabling efficient use of resources.
- **Disadvantages**:
  - **Increased Complexity**: Managing dependency packages and environment variables for functions can become complex, especially when dealing with multiple functions and different runtime environments.
  - **Environment Isolation**: Ensuring the isolation and consistency of the runtime environments of different functions is a challenge that requires additional configuration and management work.

## 7.4 Elasticsearch

Elasticsearch is a distributed, RESTful style search and analytics engine that addresses a growing number of application scenarios. It is commonly used for full-text search, log and event data analysis, real-time monitoring and data storage.

### 7.4.1 Pros and Cons

- **Pros**:
  - **Search Performance**: Elasticsearch provides efficient full-text search capabilities to quickly retrieve information in large data sets, supporting complex query syntax and highlighting.

- **Scalability**: Elasticsearch can easily scale horizontally by adding nodes to handle large-scale data. Each node can take on some of the indexing and searching tasks, thus improving overall performance and processing power.
- **Analytics**: Elasticsearch provides real-time analytics and monitoring capabilities, supports complex aggregation operations, and is able to process and analyze massive amounts of data in real time to generate valuable insights. With the help of visualization tools such as Kibana, analysis becomes more efficient.

- **Cons**:
  - **Resource Intensive**: Elasticsearch has high memory and CPU resource requirements, especially when dealing with large-scale datasets, and requires a lot of system resources to ensure performance and stability.
  - **Complexity**: Elasticsearch requires a lot of tuning and optimization work to fit specific usage scenarios. This includes index setup, query optimization, cluster management, etc., and requires a deep understanding of its internal mechanisms.

### 7.5 Jupyter Notebook

Jupyter Notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It supports various programming languages, primarily Python, and is widely used for data analysis, scientific research, and machine learning.

#### 7.5.1 Pros and Cons

- **Pros**:
  - **Interactive Computing**: Jupyter Notebooks enable interactive data exploration, allowing users to run code and see results immediately.
  - **Documented Workflow**: Combining code, results, and narrative text in a single document helps maintain a clear and documented workflow.
  - **Extensive Libraries**: Integration with a vast number of libraries (e.g., NumPy, Pandas, Matplotlib) allows for powerful data analysis and visualization.

- **Cons**:
  - **Performance**: For large-scale applications or complex computations, Jupyter Notebooks can be slower compared to traditional IDEs or scripts run in a terminal.
  - **Security Concerns**: Running untrusted notebooks can pose security risks as they can execute arbitrary code on the host machine.

## 8 Error Handling

During the completion of the entire project, we encountered numerous issues, which were largely resolved through our concerted efforts. The main problems encountered were related to the following aspects:

### 8.1 MRC

**Cinder Issues**: Experiencing frequent pod restarts (thousands) due to Cinder issues in MRC.

- **Shelve and Unshelve the Master Node**: However, this procedure is a temporary fix. The underlying issue with Cinder still persist and require future interventions. For example, after shelve and unshelve several times, Fission can survive for 30 minutes while Kafka can't survive for 10 minutes.

### 8.2 Kubernetes

**Insufficient Resources**: Kubernetes often encounters errors related to insufficient CPU or memory resources which prevent pods from being scheduled. Some solutions for this are:

- **Resource Allocation**: Increase the limits in the Kubernetes deployment configurations to ensure adequate resources are available.
- **Monitoring and Autoscaling**: Implement monitoring tools to actively track resource usage and apply autoscaling policies to adjust resources dynamically.

### 8.3 Elasticsearch

**Node Failures**: Elasticsearch clusters can experience node failures due to network issues or hardware malfunctions. Some solutions for this are:

- **Cluster Health Monitoring**: Regularly monitor the health of the cluster using Elasticsearch's built-in cat APIs to check for any signs of issues.
- **Replication and Backup**: Configure two replicas across different nodes to ensure data availability even if one or more nodes fail.

**Cluster Health is Yellow**: When the cluster health is yellow, it means that there are not enough nodes to properly distribute primary shards and replicas. This situation is less critical than a red status but still indicates that the cluster is not fully fault-tolerant. Some solutions for this are:

- **Add More Data Nodes**: Increase the number of data nodes in the cluster.
- **Ensure Node Availability**: Check if all nodes are running and can communicate with each other.

### 8.4 Fission

**Deployment Failures**: As a serverless framework, Fission's issues generally revolve around function deployment errors such as runtime execution failures. Some solutions are:

- **Logging and Tracing**: Implement comprehensive logging and tracing to capture errors during function execution. This helps in quickly pinpointing the root cause.
- **Environment Optimization**: Ensure that the environment is correctly configured for the function's requirements, including dependencies and runtime.

**Duplicate Package Installation**: When creating a Fission function that requires external packages not included in the original fission/python-env-3.9 container, such as nltk for text sentiment analysis, we must first prepare a folder containing requirements.txt and build.sh files. This folder then needs to be zipped, and a package must be created before you can deploy the function. This process can be cumbersome and slow down rapid function deployment.

- **Customize Fission Environment**: This issue can be resolved by crafting a Dockerfile that specifies and installs the required packages. You can then build a Docker image from this Dockerfile, upload it to a DockerHub repository, and utilize this repository to create Fission environments. This approach eliminates the need to create folders, zip files, and packages, streamlining the deployment process.

**Irregular Index Name**: When using the addobservations.py script to upload JSON data, it can be challenging to determine the appropriate Elasticsearch index for storing the data since the script is reused for uploading various types of data to different indexes.

- **Add Index Name as Argument**: This issue can be resolved by adding the index name to the request body as an additional argument when packaging data.

### 8.5 Jupyter Notebook

**Kernel Issues**: Jupyter notebooks can crash if the kernel is overloaded or encounters an unexpected error. Some solutions are:

- **Kernel Management**: Regularly update the Jupyter environment and the kernels to the latest versions to minimize compatibility issues.
- **Resource Limits**: Set resource limits for the Jupyter kernel to prevent it from consuming excessive memory or CPU.

## 9 Limitations and Future Improvements

### 9.1 Limitations

- **Resource Scalability in Melbourne Resource Cloud**: The cloud infrastructure occasionally struggled under load, particularly when scaling up resources during data-intensive operations. This limitation impacted the project's ability to process large datasets in real time.
- **Kubernetes Resource Allocation**: Managing resources within Kubernetes posed challenges, particularly with pods requiring high resource allocation which led to scheduling issues. This was compounded by inefficient load balancing and auto-scaling configurations.
- **Elasticsearch Query Performance**: As the project scaled, Elasticsearch experienced delays in query processing due to the complexity and volume of the data being analyzed. This affected the speed and efficiency of retrieving insights from social media data.
- **Write Constraints and Master Node Failover in Elasticsearch**: Elasticsearch restricts write request directed to an index to only one node at a time (the node hosting the primary shard), It has elections in place to establish another master node in case one fails, and there is the possibility of not reaching a minimum number of master-eligible nodes (a quorum), hence forcing the cluster to stop accepting write requests.

- **Fission Cold Start Delays**: The serverless framework Fission used in the project exhibited significant latency during the initialization of functions after idle periods, which led to delays in data processing workflows.
- **Data Integration in Jupyter Notebook**: Integrating and synchronizing data from various sources via Jupyter Notebook presented challenges, particularly in terms of real-time data handling and consistency across the analytical models.
- **Lack of geometry information of Mastodon**: Without latitude and longitude coordinates or other geographic identifiers, we cannot directly plot the data on maps. This makes it challenging to visualize the spatial distribution of posts, users, or any geographic patterns within the data.

### 9.2 Future Improvements

- **Enhanced Cloud Scalability**: Adopting an auto-scaling solution that dynamically adjusts to the workload demands in real time could mitigate the resource limitation issues experienced in the Melbourne Resource Cloud.
- **Optimized Kubernetes Configurations**: Implementing advanced Kubernetes configurations such as better resource quotas and improved load balancing strategies could enhance resource management efficiency.
- **Elasticsearch Performance Tuning**: Fine-tuning Elasticsearch configurations, such as optimizing index management and enhancing query caching mechanisms, could significantly improve response times and data processing capabilities.
- **Reducing Fission Cold Starts**: Implementing a pre-warming solution for Fission functions or exploring alternative serverless frameworks that offer faster initialization times could reduce the impact of cold starts.
- **Robust Data Integration Tools**: Integrating a more advanced data management platform that offers better real-time processing and data synchronization capabilities could resolve the issues faced with Jupyter Notebook.
- **Continuous Monitoring and Adaptive Strategies**: Establishing a continuous monitoring framework to track system performance and adapt configurations in real time could prevent potential bottlenecks and improve system resilience.

These enhancements aim to build a more robust and efficient system for analyzing the impact of technological advancements on social media perceptions, ensuring that future projects can handle complex analyses more effectively.

## 10 Conclusion

In summary, our application utilizes data from platforms such as Twitter, Mastodon, and the SUDO platform to explore the interplay between social media activity and technological advancement in Australia. Leveraging cloud-based solutions provided by the Melbourne Research Cloud (MRC), we effectively gather, store, and analyze large datasets. This method overcomes limitations inherent in SUDO, such as the lack of real-time social media data exploration and the inability to consolidate

data from diverse sources for comparative analysis. Our comprehensive cloud solution encompasses a complete ecosystem, including backend servers, frontend visualization, an Elasticsearch database, and data processing and analysis modules. All components are containerized using Docker, showcasing automation and scalability through Kubernetes.

Through a detailed analysis of four scenarios, we have observed that while technological progress has had many beneficial impacts on society, there remains a significant number of individuals who hold pessimistic or neutral views towards technological development. On one hand, the data indicates that while technologies like AI and 5G are commendable, greater consideration must be given to potential drawbacks during their implementation. On the other hand, it appears that most people are not deeply concerned with the intricacies behind these technological advances and do not have a definitive stance on technological progress.

## References


[1] U. of Melbourne, "Melbourne research cloud documentation," 2024. [Online]. Available: https://docs.cloud.unimelb.edu.au/

[2] M. Servers, "Mastodon server list," 2024. [Online]. Available: https://mastodonservers.net/

[3] U. of Melbourne, "University of melbourne eresearch services (sudo)," 2024. [Online]. Available: https://sudo.eresearch.unimelb.edu.au/

[4] A. B. of Statistics, "Australian statistical geography standard (asgs) edition 3, digital boundary files," 2021. [Online]. Available: https://www.abs.gov.au/statistics/standards/australian-statistical-geography-standard-asgs-edition-3/jul2021-jun2026/access-and-downloads/digital-boundary-files

[5] A. Competition and C. C. (ACCC), "Mobile infrastructure report 2023," 2023. [Online]. Available: https://www.accc.gov.au/by-industry/telecommunications-and-internet/mobile-services-regulation/mobile-infrastructure-report/mobile-infrastructure-report-2023